IBM WebSphere Developer Technical Journal: A guided tour of WebSphere Integration Developer -- Part 1

Get a driver's view of the WebSphere Integration Developer landscape

Greg Adams (Greg_Adams@ca.ibm.com), Distinguished Engineer, IBM
Richard Gregory (gregoryr@ca.ibm.com), Staff Software Developer, IBM
Jane Fung (jcyfung@ca.ibm.com), Advisory Software Developer, IBM
Randy Giffen (Randy_Giffen@ca.ibm.com), Advisory Software Developer, IBM

Summary: This article is the first in a series exploring a service-oriented approach to application integration using IBM® WebSphere® Integration Developer. This first article provides an overview of WebSphere Integration Developer and its key components and concepts.

Date: 22 Feb 2006
Level: Introductory

From the IBM WebSphere Developer Technical Journal.

Introduction

This article series gives you a guided tour of developing applications with WebSphere Integration Developer. This first article gives you a broad overview of WebSphere Integration Developer and its main concepts.

Subsequent articles will dive into each concept and the related construction tools. We'll look at each area of the product, why it matters, what you can do with it, and lastly give you the driver's seat to build the next piece of an overall application. Some topics that we'll cover in future articles include:

- SOA development
- Building and assembling a simple application
- Business processes, state machines, and rules
- Human tasks
- EIS connectivity support
- Mediation and selectors

Although these articles build on each other as a series, each stands on its own if you want to delve into a particular area of interest.

What is WebSphere Integration Developer?

You are probably wondering what WebSphere Integration Developer is, and why you should care. Companies today face increasing pressure to integrate parts of their enterprise, automate their systems and provide new channels for their customers. Companies need products and solutions that are flexible and based on standards.

Some of the problems typically encountered in the integration arena include:

- Synchronizing data among two or more heterogeneous enterprise information systems (EIS).
- Intelligently brokering product requests from consumers to multiple producers.
- Publishing product data to a global repository, thus enabling consumers to access and use the information. Publishing can range from making catalogs of products available, to participating in a global online marketplace.
- Orchestrating multiple existing business processes using an overarching process.
- Managing order processing from order receipt to both inventory management and supply chain management.
- Assigning, approving, and escalating tasks to effectively deal with customer requests.
- Dynamically responding to changing business conditions by altering the rules and decisions governing the business.

WebSphere Integration Developer tackles these and other types of application integration problems. From the ground up, WebSphere Integration Developer is based on industry standards (notably WSDL, XSD, BPEL, Java™, and UML) and is on the leading edge of evolving standards (the Tuscany Service Component Architecture is a good example). To build applications on these standards you use a set of visual construction tools and higher level concepts, which lets you focus on the business problem, and not have to write masses of J2EE code or be a WSDL rocket scientist. You get the standards without having to eat, sleep, and breathe them.

From a WebSphere Integration Developer perspective, a service-oriented architecture means you can focus on the key components of your system, build them visually, connect them together visually and then be off and running using WebSphere Process Server. Afterwards, you can also visually unit test and debug your entire application or individual parts of it.

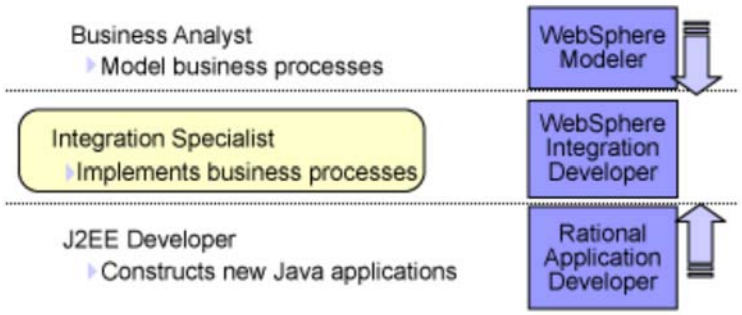
WebSphere Integration Developer supports top-down, bottom-up, and meet-in-the-middle construction. You can start at the top--the design level, lay out your overall vision and then gradually drill down and implement the components (services). Alternatively, you can work bottom up, implement the services, and then compose them together into a larger application. More likely, you will find yourself using a meet-in-the-middle approach, perhaps laying out the initial high-level vision, then use the Enterprise Metadata Discovery utility to explore an enterprise information system, and define services that connect to it. You might also want to pull in and reuse an external Web service that one of your business partners has provided.

Who uses WebSphere Integration Developer?

Perhaps the real question we should be asking is, *what roles do users of WebSphere Integration Developer fulfill,* and *when, during the overall development process, do they use the tools?*

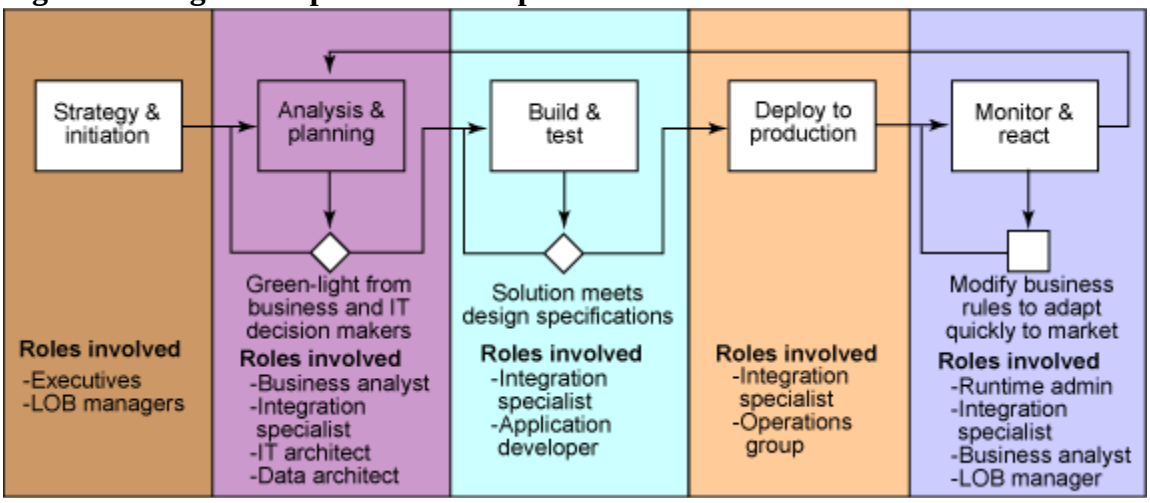
WebSphere Integration Developer targets *integration specialists*. These users are not Java, WSDL, or XSD experts. They focus on integrating applications and addressing the types of problems discussed above -- and of course they want to do it as easily as possible. Figure 1 illustrates the relationship between the skill set of the integration specialist and other user roles.

Figure 1. WebSphere Integration Developer users



The process of application development involves many user roles at different stages of development. Note that by *role*, we refer to a capacity in which an individual works; any given person could actually fulfill multiple roles in a company. Figure 2 illustrates the portions of the software development process in which the integration specialist is typically involved. The integration specialist takes over where the business analyst leaves off, developing the integration application, testing and debugging it, and then ultimately, when all the t's are crossed and the i's dotted, deploying it to a production server.

Figure 2. Integration specialist development involvement



A tour

It's time to warm up the car and take a tour of WebSphere Integration Developer's key concepts and tools. In subsequent articles, we will focus on the importance and value of each element and show you how to use the concept or tool as part of building a sample application. For now we'll just do a quick cruise by each item.

- Pick your route - service implementation types
  - Business processes
  - Business state machines
  - Business rules and decision tables
  - Selectors
  - Interface maps and business object maps
  - Human tasks
  - Web services
  - Enterprise information system services

Pick your route -- service implementation types

Services are the main building blocks of a service-oriented architecture. WebSphere Integration Developer includes a diverse set of service-building tools that cover the spectrum of problems you might encounter. For the most part, you can build the service implementation types mentioned above using visual construction tools, which is why you don't need to be an expert programmer to develop applications with WebSphere Integration Developer. Most services are built using visual construction tools, with the one exception being a Java or EJB service. You can use these Java and EJB services to leverage existing Java applications, or, if your organization has some Java heads, to build Java services using IBM Rational® Application Developer (see Want to learn more about related products?).

You can implement your services using a variety of programming paradigms, from process-flow style BPEL processes, to state machine-style event management, to declarative business rules style. The style of implementation you select will be determined both by your comfort level with a given paradigm as well as the nature of the problem. For example, some problems lend themselves to more succinct expression as a state machine than as a set of declarative rules. Let's have a look at these different service types and discuss when you might want to use one versus the other.

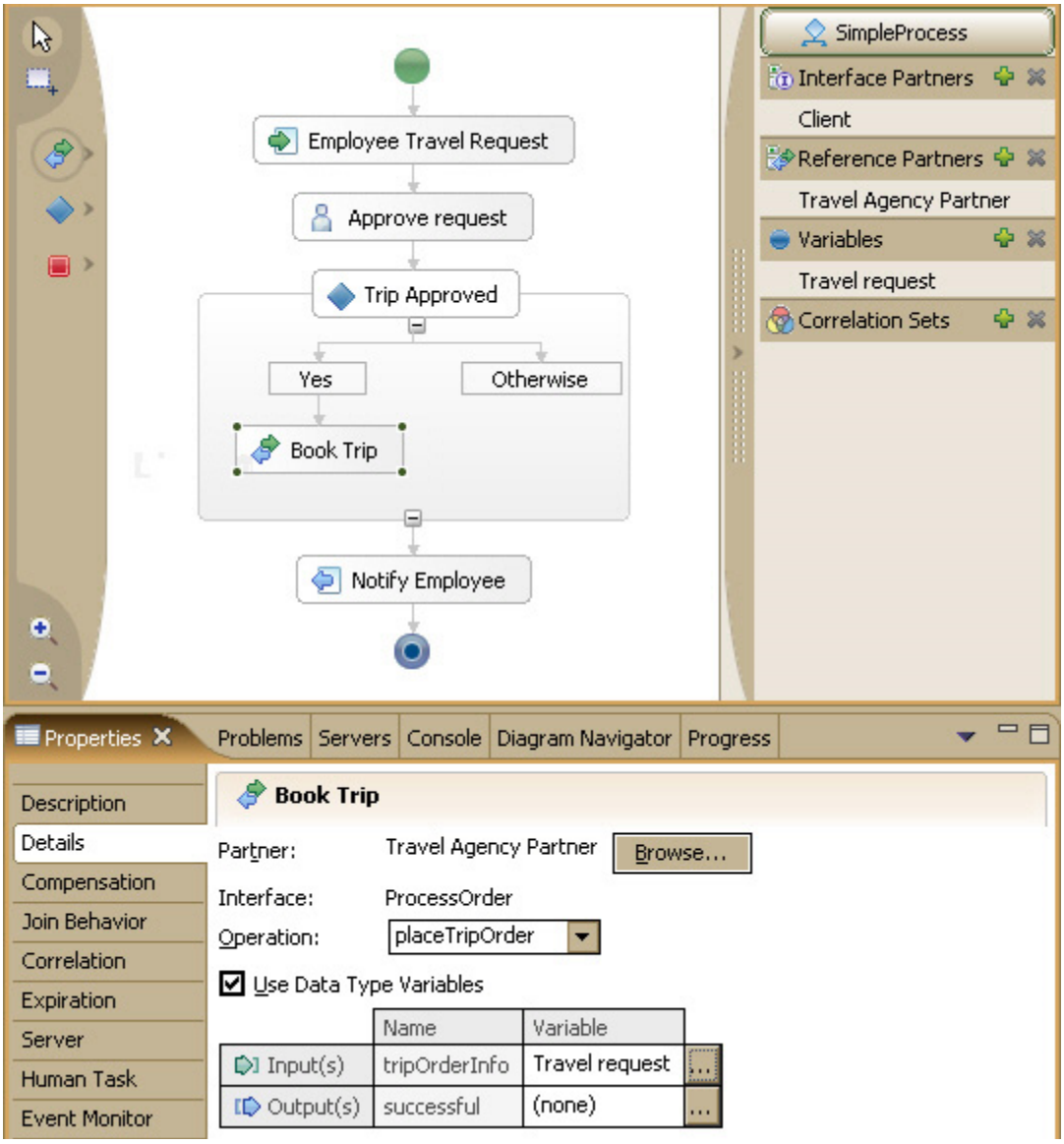
Business processes

A *business process* provides the primary means for coordinating enterprise services and describing your business logic. A business process consists of a series of activities or steps that are executed in a specific order, sequentially or in parallel. The business process editor is a visual construction tool that allows you to quickly author a business process based on the BPEL standard.

A business process itself is a service. You can use it to coordinate reusable subprocesses or other services, which may have any other implementation type. An important aspect of business processes is their longevity and interaction with people. A process can be very short lived; this is common in highly automated systems. Alternatively, the process could be extremely long running, perhaps days or even months, and may wait on a human user to complete a specific work item associated with an activity before the process can continue. For example, the business process could be patiently waiting for a manager to approve a travel request.

Figure 3 shows how to construct a simple travel booking business process with the business process editor.

Figure 3: The business process editor



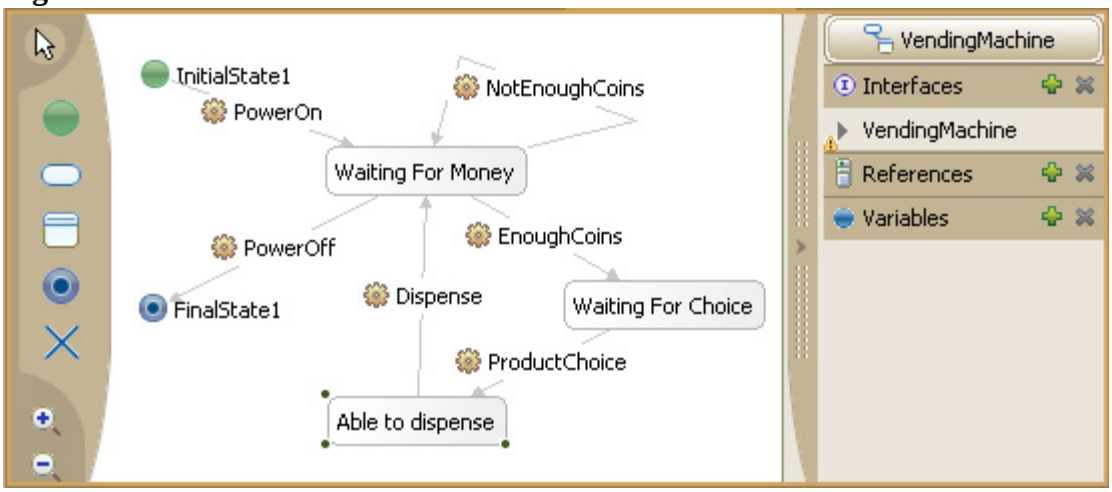
Business state machines

A *business state machine* is an event-driven business transaction that defines a set of states for a given part of the application. The state machine moves from one valid state to the next based on external events it receives. For a given event, you use conditions to determine the new valid state. A simple illustration is a vending machine; when it receives enough money, it transitions into a state where the selection buttons become active. When you make a selection, it changes to a state where it can dispense the merchandise (let's say a quadruple chocolate candy bar). You construct business state machines using the state machine editor, which, like the business process editor, is a visual editing tool that requires little or no Java programming experience.

You can use both business state machines and business processes to coordinate parts of your application. There are some subtle differences between them, which makes one more appropriate for certain types of problems than others. State machines are well suited to cyclical patterns or cases where you "naturally" think in a set of valid states. This is important because in a state machine you actually are not doing anything in a state, you are simply waiting to be told to pick up your gear and go to the next state. While the state machine is racing to get into the next state, it can do some work, such as handing the customer their chocolate bar in the vending machine example. Business processes, in contrast, do their work in the activities. They are well suited to sequential and parallel tasks. Just as with business processes, business state machines can invoke other service implementation types and can themselves be invoked as a service. Figure 4 shows a business

state machine that we constructed using the business state machine editor.

Figure 4: The business state machine editor



For those who are familiar with UML, a business state machine is a subset of a UML state machine and is more appropriate for business users.

Business rules

A *business rule* captures and implements business policies and practices. A rule can enforce business policy, make a decision, or infer new data from existing data. It can be specified using two different formats: a rule set, or a decision table.

That explanation sounded too much like a collection of buzzwords, so an example is in order. A business rule is something like, *if a customer is a gold customer and has been with us for ten years then give them a 10% discount.* This business rule is a simple *if-then* rule. If the rule evaluates to true, it performs an *action*, which in this case, is giving the customer a discount. Business *rule sets* consist of groups of rules that allow great flexibility in the implementation of complex business logic.

A *decision table* is designed to handle basic business rule logic. It does not have the same flexibility as a rule set; however, it offers great simplicity for capturing simple rule logic in a table format. A classic example of a decision table is one that frequent travelers are familiar with. Suppose you want to escape the cold and use up your frequent flyer points to go Hawaii. You look at a table and locate the row and column that correspond to your home city and Hawaii respectively; where they meet shows how many points are required for that trip. This is just one kind of business rule logic that can be easily captured in a decision table.

This brings us to a key aspect of a business rule: how dynamic it is, or, its ability to respond to a changing business environment. You can use business rules to modify key business parameters on a production server dynamically and have them take effect immediately. For example, suppose it is an unusually warm winter and there is little demand for tropical destinations. You decide to reduce the frequent flyer points needed for Hawaii. It is trivial to do this at runtime by viewing the table and changing the value.

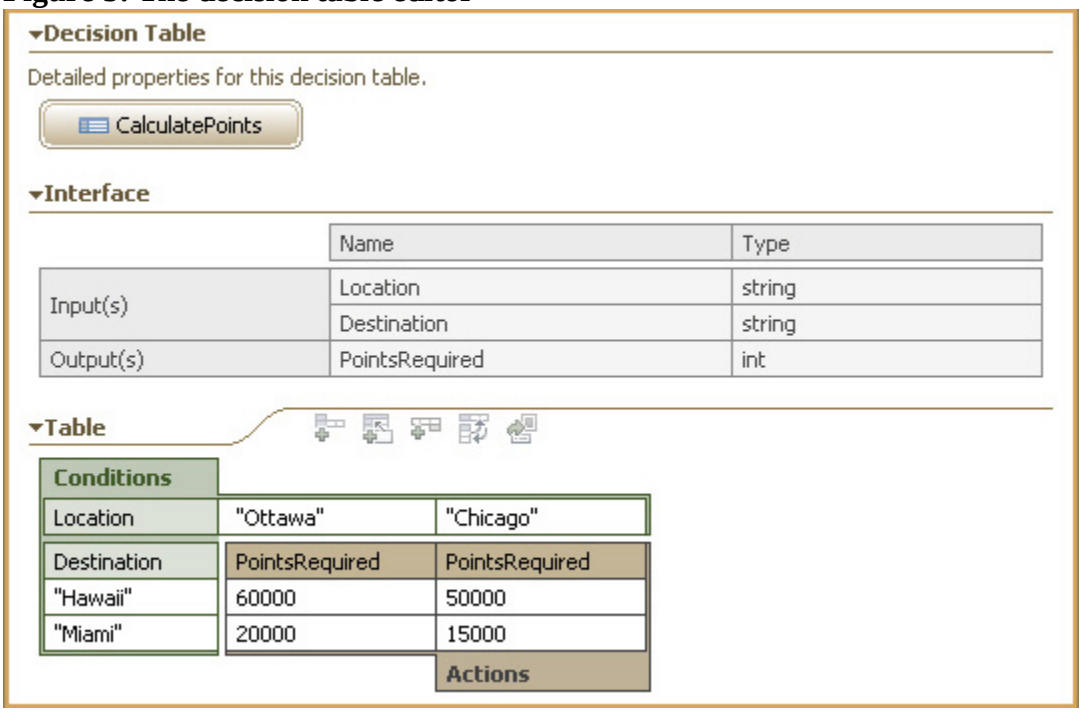
Later in this article, we will briefly visit WebSphere Business Monitor; you can use this product to monitor your business in real time so that you can react to key observations and conditions by adjusting your business rule policies.

In general, you should use business rules to make a decision when any of the following conditions apply:

- You want to change the results at runtime, on a running server
- The decision itself naturally takes on the form of a table
- The decision itself naturally takes on the form of a series of simple choices--easily thought of as if-then statements

Figure 5 shows how we would create a decision table using the decision table editor.

Figure 5: The decision table editor



Selectors

Selectors are an easy way to respond to a service request and route it to another service that is designed to handle it. The routing can vary over time. You use a selector to invoke different service implementations based on a date. For example, let's say you want your online Christmas card business to use the regular sales service until December 25th, and then use the mega-discount service afterwards. You can use the visual selector editor to build a selector that will intercept sales service requests and select the regular service up until Christmas, and afterwards, select the discount service. Figure 6 shows how we would create such a selector.

As with business rules, you can change selectors dynamically on a production server. You can modify the service destinations and the date selections to suit your evolving business needs.

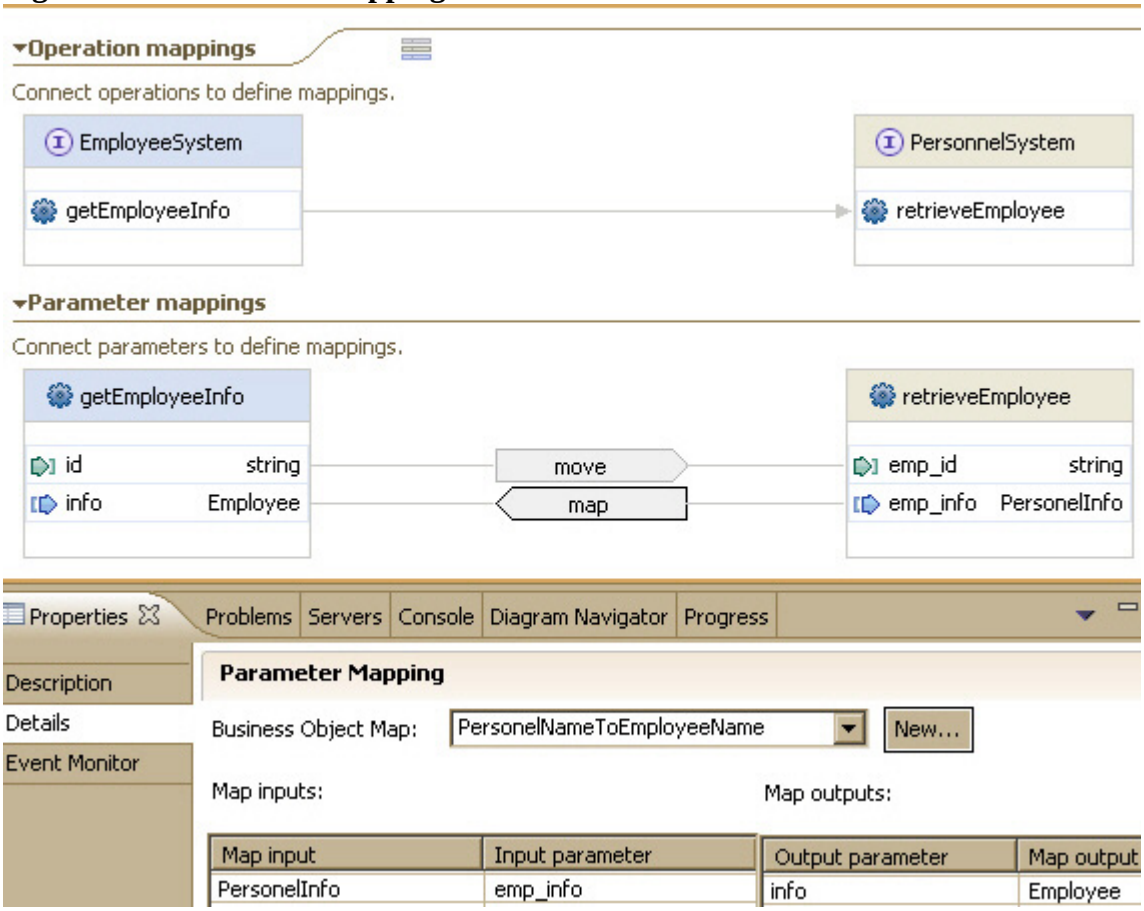
Figure 6: The selector editor

Interface maps

Sometimes you might have two services that just will not to speak to each other because they do not understand the same set of operations. While you might be inclined to get grumpy at this dilemma, a simple solution is to use an interface map. An *interface map* describes how an operation of one service translates into another. We will discuss interfaces in Putting it together.

When mapping two interfaces, first map their operations and then map the input and output messages. If the input and output messages do not have the same type, you would use a *data map* (explained in the next section) to map between the two types. Figure 7 shows how we would construct an interface map using the interface mapping editor.

Figure 7: The interface mapping editor



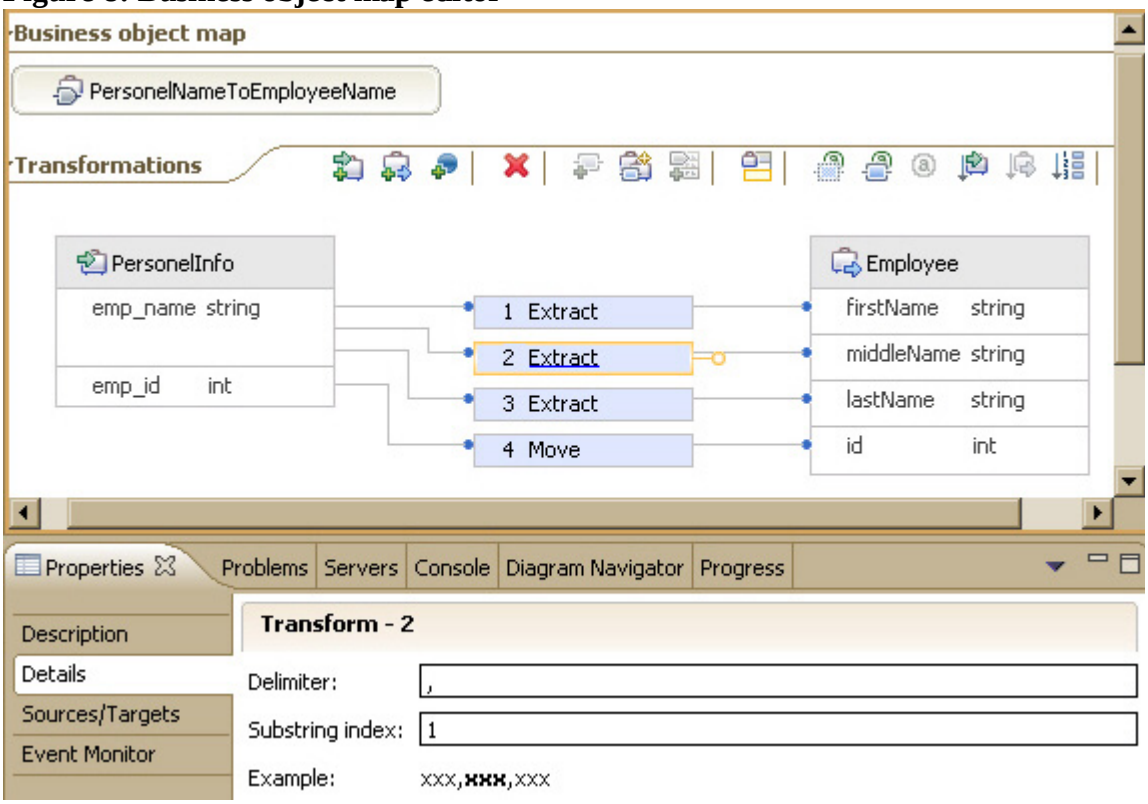
Business object maps

Business object maps, also known as *data maps*, are used for translating business data from one type to another. It is often necessary to map one business object (see Key landmarks -- business objects) into another when coordinating heterogeneous systems, or even as part of normal business logic.

Using the business object map editor, you can graphically create a map that transforms a business object and its fields to another business object as Figure 8 shows. As a simple example, consider a name from an employee information service that needs to be passed to your travel booking service. Suppose the service is hosted by an old clunky system that returns a comma delimited string, yet the travel booking service requires three separate fields (first/last/middle name). The mapping process in this case means taking the full name and separating it into the required pieces. The editor provides quick convenience mechanisms for all the common maps you will need (concatenation, for instance), and provides a way for you to define your own custom conversions using either Java or a visual snippet (more on those later).

Business object maps support 1-to-*n*, *m*-to-1 and *m*-to-*n* mappings among business objects.

Figure 8: Business object map editor

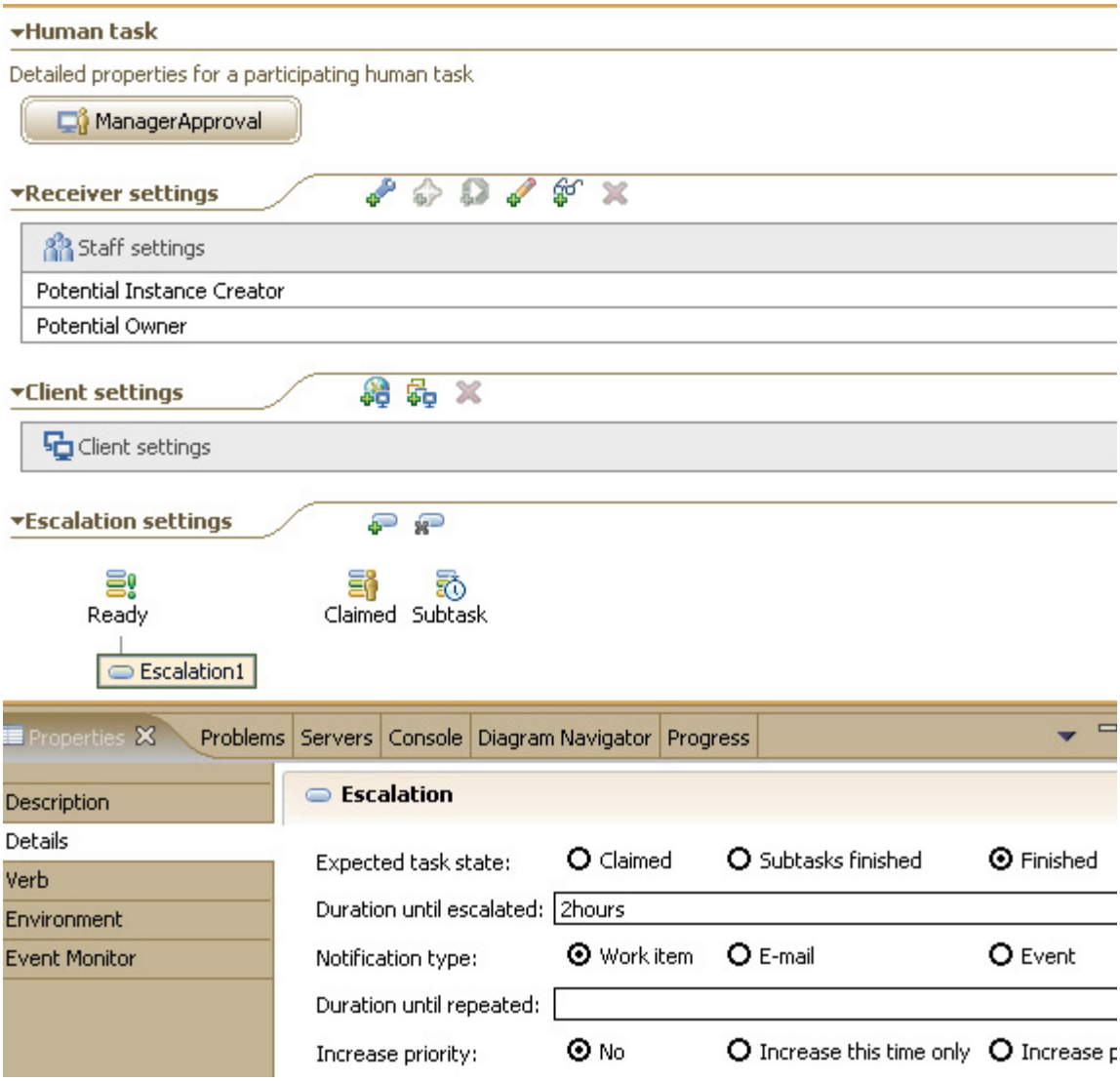


Human tasks

A human task is, quite simply, a unit of work done by a human. Quite often, this type of task involves interactions with other services, and thus becomes a task within a larger business goal. You can use WebSphere Integration Developer and WebSphere Process Server to escalate or delegate human tasks in the event they are not handled in a timely fashion. Tasks can be assigned to individuals or groups of individuals (any manager, for example) based upon your organization structure as defined in a system such as LDAP.

You can create human tasks using the visual human task editor as shown in Figure 9.

Figure 9: The human task editor



Web services

The services we have mentioned are first-class services, but so far are only accessible to other WebSphere applications. However, you can easily expose any of the service types listed above as Web services. We will explain this in the Assembly diagrams section. You can even go one step further and create standard Web services using the capabilities inherited in WebSphere Integration Developer from Rational Application Developer. You will find more information in Want to learn more about related products and in the references at the end of this article.

Enterprise information system (EIS) services

Since your company likely depends on at least one EIS, you can easily turn its applications into services. You do this using the Enterprise Service Discovery wizard, which uses standard J2C resource adapters to connect to and query back-end systems such as CICS® or PeopleSoft. There are two resource adapters that ship with WebSphere Integration Developer, one for CICS ECI and one for IMS™.

Figure 10 shows how you would create an operation for a purchase-order service, based on the data it found on the PeopleSoft server. When you complete the steps for the wizard, you have a service that lets you access your EIS the same way you access any other service.

Figure 10: The Enterprise Service Discovery wizard

Java and EJB services

As we mentioned, if your company has some seasoned Java developers, you can still create or reuse plain old Java objects or EJBs as your service implementations. Calling other services from your Java code is as easy as calling them using the visual editors. If you were to use Jave code to make the same service call to the Travel Agency partner as in the Business process section (recall Figure 3 shows the book trip activity making this call), the code would look something like this:

```
result = locateService_TravelAgencyPartner().placeTripOrder(travelRequest);
```

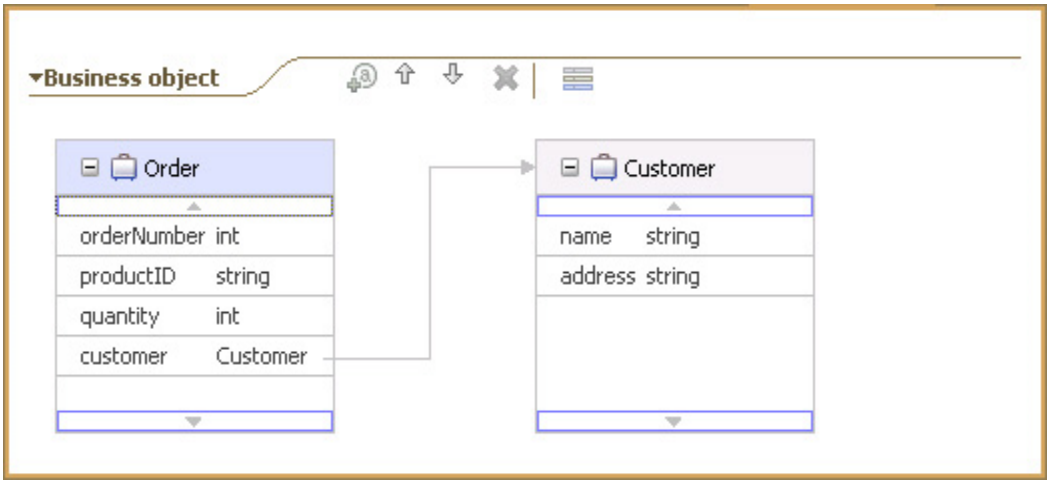
The service call is made simple by helper code that is generated for each reference (you create the references in an assembly diagram, as we will explain shortly).

Key landmarks -- business objects

Back in the section called Business object maps, we showed how to map variations of a business object so that one service can understand another. In this section we'll give you an idea of what business objects are. A *business object* is the primary currency of your business application. Examples of business objects include a customer order, a customer, and items in your inventory. A business object consists of a set of fields and their values. A field may in turn be another business object. For example, an Order business object may have a customer field which in turn is a Customer business object.

Business objects are created using the business object editor, which provides a graphical view of your business objects. You can also use the editor to work with the object in a tabular format, if you prefer to use the keyboard for quickly filling in forms.

Figure 11: The business object editor

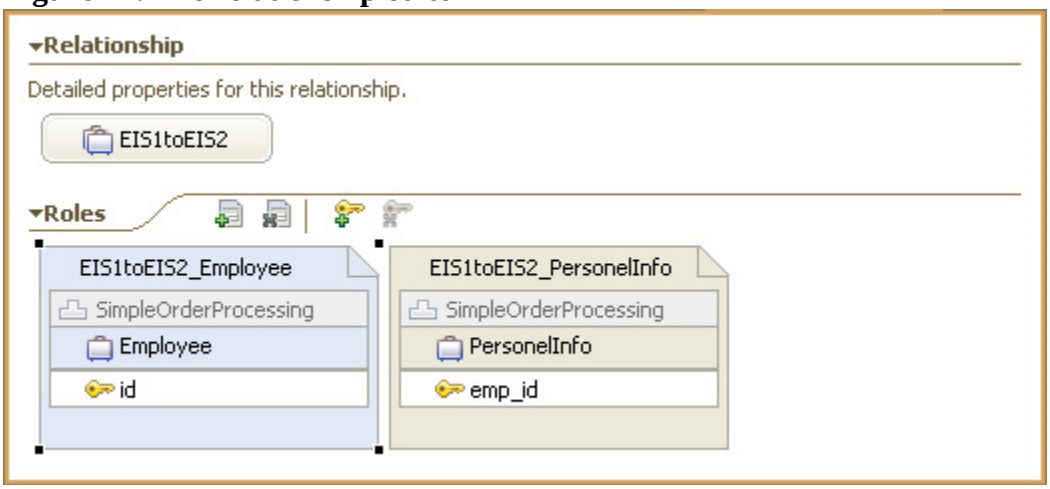


Relationships

A *relationship* correlates at least two semantically equivalent business objects that are represented in different physical formats. Suppose EIS1 had employees whose name was represented as a full name, and EIS2 also had employees, but with the name stored as a first name and a last name. If these are really the same employees, then you would use a relationship to indicate this. Now, if the information is updated for an employee in one EIS, it can easily be updated for that employee in the other. By establishing a relationship between them, you are saying it is in fact the *exact same* employee.

Relationships are created using the visual relationship editor, as shown in Figure 12.

Figure 12: The relationship editor

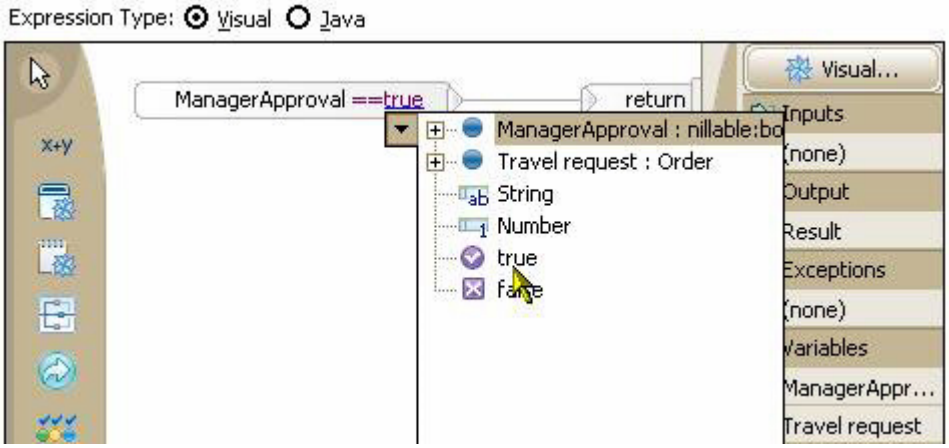


Visual snippets

There are times in your business processes, state machines, or other services where you might find yourself needing to write some nitty-gritty logic. You can use WebSphere Integration Developer to do custom code in a number of places, and although you are free to throw in some Java, a popular alternative is to use the visual snippet editor. This editor lets you describe a more detailed level of logic without having to drop down to textual Java.

Figure 13 shows a simple way to determine which branch should execute in the choice following the human task shown previously in Figure 9.

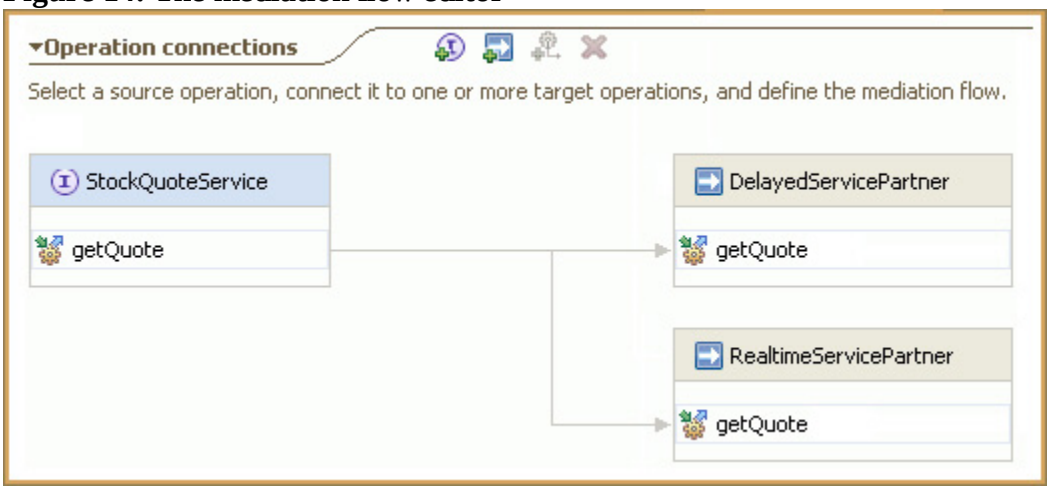
Figure 13: Visual snippet editor



Mediation services

A *mediation service* intercepts and modifies messages that are passed between existing services and their clients. Mediation services are implemented using mediation modules that contain *mediation flows*. For example, suppose you had two types of customers using your stock quotation service: regular (non-paying) and gold. Your service delegates to two external services to fetch the stock prices: one provides delayed quotes, and one provides real-time quotes. You could use a mediation flow to route gold member requests from your service to the real-time quotation service, and the rest to the delayed service. Figure 14 shows how you would use the meditation flow editor to create such a mediation.

Figure 14: The mediation flow editor



Putting it together

Now that you are familiar with the various ways to implement a service, we will show you how those fit together in a deployable integration application.

Modules and components

Any service that you create becomes a *component* that you can drop into an assembly diagram (see the next section). The components are grouped together into a *module*, which can be deployed to WebSphere Process Server (more on that in a moment).

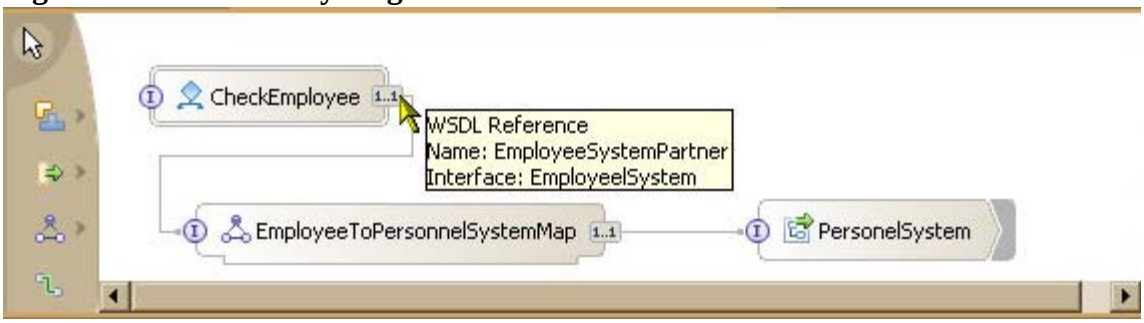
An important aspect of components is their interfaces. An *interface* lets others know how to talk to your service, and are defined using WSDL or Java. So, a component requires an interface and the implementation must adhere to that interface. You can wire any reference of one component to any other component, provided the component's interface matches the interface required by th reference.

As we mentioned, you might not define the implementations for your services if they already exist, such as a Web service (which has WSDL interfaces) or an EIS application (the Service Discovery wizard creates an appropriate interface). In this case, you use an *import*, which you can reference in the same manner as any other component. The fact that it is implemented externally is transparent; you just use the reference. Similarly, when you expose your services outside WebSphere or to other applications within WebSphere you just add an *export* with the appropriate interface and wire it to the component you want to expose. You assign bindings to imports and exports whose properties determine how the service is accessed. For example, if your import has a Web service binding, you will specify an endpoint and a port.

Assembly diagrams

You use an *assembly diagram* to wire your service components together within a module. Figure 15 shows the assembly diagram using EmployeeToPersonnelSystemMap. Here you have a CheckEmployee process that expects to use the Employee EIS. Using the map, you can keep that reference (and, more importantly, the business logic that uses it) and wire it to the PersonnelSystem using the mediation. The EmployeeSystemPartner reference displayed on the CheckEmployee component corresponds to the same reference within the process.

Figure 15: The assembly diagram editor



Want to learn more about related products?

WebSphere Integration Developer complements several other IBM WebSphere business integration products to assist in the development of service-oriented applications. The most important of these are WebSphere Business Modeler, WebSphere Process Server, and WebSphere Business Monitor. Let's have a brief look at each of these products and how WebSphere Integration Developer fits in with them.

WebSphere Business Modeler

Although you can begin building your service-oriented applications using WebSphere Integration Developer directly, it may be more useful to first model your business processes using WebSphere Business Modeler. Modeling your business processes with WebSphere Business Modeler helps you gain a better understanding of your business, validate enhancements and transformations, and discover potential areas for process improvement, or latent value in existing processes, before implementing technical solutions.

Aside from helping to provide a starting point for implementing solutions, WebSphere Business Modeler also provides other benefits, such as giving you the necessary information for process compliance, documentation

and training, and allowing you to run simulations to reveal the implications of process modifications (the simulation tools are included with the advanced version of the product). You can define company resources such as existing information systems, equipment, or employees and business items such as invoices and documents with WebSphere Business Modeler and incorporate them in the process models. At a higher level, you can use WebSphere Business Modeler to model the relationships and interactions between the various entities within an enterprise.

Just as WebSphere Integration Developer can be used by people with varying skill sets, so can WebSphere Business Modeler. WebSphere Business Modeler allows the people who know the business to do the modeling before beginning implementation. For example, a business analyst who needs to model processes at a higher-level can do so using the basic mode, while the more technically experienced person can use the intermediate or advanced user modes to specify deeper process details or more complex business logic.

When you have your business processes modeled, you can pass the models to a development platform for implementation. You can use WebSphere Integration Developer to import the models, and use them to build and test a complete set of SOA applications.

Rational Application Developer

WebSphere Integration Developer is built on Rational Application Developer. Rational Application Developer is a comprehensive development environment designed to meet a variety of development needs, including simple Java applications, sophisticated Web interfaces and portals, and EJB and data access components. It, in turn, is built on Eclipse, which is an open-source platform for development and creating application development tools.

So, when you need to develop portal or Web client applications for your services, you can enable additional features of WebSphere Integration Developer that are found in the underlying Rational Application Developer tool suite. You can also use the Rational Application Developer tools to expose your services as Web services. In fact, you can do additional development for your applications at the J2EE level.

WebSphere Process Server

After you complete the implementation phase, you use WebSphere Integration Developer to deploy the applications to WebSphere Process Server. WebSphere Process Server is a service-oriented architecture integration platform based on WebSphere Application Server. It has evolved from proven business integration concepts, application server technologies, and the latest open standards. It plays a key role in helping your business cope with a changing environment by providing execution-time support for the various types of service components contained within an application developed with WebSphere Integration Developer.

While you can deploy standard Business Process Execution Language (BPEL)-based applications to any BPEL-compliant application server, WebSphere Integration Developer provides the development tools that let you take advantage of WebSphere Process Server's additional capabilities. These capabilities include human tasks to support the manual parts of a business process, business state machines for event-driven processes, business rules to facilitate on-demand changes in a process, and selectors that enable dynamic business processes.

WebSphere Process Server is powered by WebSphere Enterprise Service Bus which, in addition to enabling execution of a true service-oriented architecture, provides J2EE Connector architecture (JCA) and Web services connectivity as well as Java Message Service (JMS) messaging support.

WebSphere Business Monitor

When your applications are up and running on WebSphere Process Server, you will need to capture

information about them to help you identify problems, discover faults that have occurred, and determine whether any process modifications are necessary. This area is where WebSphere Business Monitor comes in handy. WebSphere Business Monitor collects events emitted by WebSphere Process Server using the Common Event Infrastructure. From them, it calculates Key Performance Indicators (KPIs) and metrics based on a business measures model.

Using the process models you created, you create a business measures model using the business measures editor that comes with WebSphere Business Modeler. These models define measurements and measuring points, event filters and correlations, and the sources of business data that you want to monitor. The monitor server uses these models as it receives and processes events from the server.

The dashboard client component of WebSphere Business Monitor enables users to monitor business performance using a customized set of views. The views show different representations of active process instances and their status, reports, KPIs, and KPI scorecards, and alerts based on the business measures model. The adaptive action manager can notify you by way of the dashboard, as well as by e-mail, pager, or cell phone based on criteria that you specify.

Finally, WebSphere Business Monitor lets you export monitored values, which you can then import into WebSphere Business Modeler for continuous improvement of the business processes.

Summary

We've taken a quick tour of WebSphere Integration Developer and some of its key concepts. As you can see, the environment is highly visual, with an emphasis on the higher-level concepts of your application. This experience is often described as *point*, *click*, and *integrate*.

In future articles we will get into the details of each concept and build an example around it. Through the series you will see a complete integration application take shape. Look for Part 2 in next month's WebSphere Technical Journal, which will cover SOA development in WebSphere Integration Developer, including how to build a simple SOA application.

More articles in this series

- Part 2: SOA development with WebSphere Integration Developer
- Part 3: Building a simple service-oriented application

Resources

Learn

- A guided tour of WebSphere Integration Developer -- Part 2: SOA development with WebSphere Integration Developer
- developerWorks: WebSphere Process Server and WebSphere Integration Developer resources
- WebSphere Process Server product information
- WebSphere Integration Developer product information

- WebSphere Enterprise Service Bus product information
- WebSphere Business Modeler product information
- WebSphere Business Monitor product information
- Rational Application Developer product information
- WebSphere Process Server: IBM's new foundation for SOA
- Build a Hello World SOA application
- Service Component Architecture
- Business Process Execution Language (BPEL)
- Web Services
- Common Event Infrastructure
- Tuscany Service Component Architecture
- developerWorks: WebSphere Business Integration zone
- developerWorks: WebSphere development tools zone

Get products and technologies

- Download the WebSphere Business Modeler trial
- Download the Rational Application Developer trial

Discuss

- WebSphere Integration Developer forum
- developerWorks blogs

About the authors

Greg Adams was the lead architect for the user interface in the award-winning Eclipse platform, and more recently, has been the lead architect and development lead in the core WebSphere Business Integration Tools, including WebSphere Studio Application Developer Integration Edition and WebSphere Integration

Developer. Greg led the delivery of IBM's first complete services oriented architecture (SOA) tools stack and the first BPEL4WS standards supporting Business Process Editor; both critical deliverables in support of IBM's On Demand strategy.

Richard Gregory is a software developer at the IBM Toronto Lab on the WebSphere Integration Developer team. His responsibilities include working on the evolution and delivery of test tools for WebSphere Integration Developer.

Jane Fung is an Advisory Software Developer at IBM Canada Ltd, where she is responsible for developing the Business Process Execution Language (BPEL) and Business Rules debuggers in WebSphere Integration Developer. Prior to that, she was the team lead of the WebSphere Studio Technical Support team.

Randy Giffen is an advisory software developer at the IBM Ottawa Lab on the WebSphere Integration Developer team. He was responsible for WebSphere Integration Developer's business state machine tools and the visual snippet editor. Prior to to this he was a member of user interface teams for WebSphere Studio Application Developer Integration Edition, Eclipse, and VisualAge for Java.